# Extending the BT NAS Parallel Benchmark to Exascale Computing

Rob F. Van der Wijngaart
Intel Corporation
3600 Juliette Lane
Santa Clara, CA 95054
Email: rob.f.van.der.wijngaart@intel.com

Srinivas Sridharan
Intel Corporation
23-56P, Outer Ring Road
Bangalore, India 560017
Email: srinivas.sridharan@intel.com

Victor W. Lee
Intel Corporation
3600 Juliette Lane
Santa Clara, CA 95054
Email: victor.w.lee@intel.com

***Abstract*—The *NAS Parallel Benchmarks* (NPB) are a well-known suite of benchmarks that proxy scientific computing applications. They specify several problem sizes that represent how such applications may run on different sizes of HPC systems. However, even the largest problem (class F) is still far too small to exercise properly a petascale supercomputer. Our work shows how one may scale the *Block Tridiagonal* (BT) NPB from today's published size to petascale and exascale computing systems. In this paper we discuss the pros and cons of various ways of scaling. We discuss how scaling BT would impact computation, memory access, and communications, and highlight the expected bottleneck, which turns out to be not memory or communication bandwidth, but network latency. Two complementary ways are presented to overcome latency obstacles. We also describe a practical method to gather approximate performance data for BT at exascale on actual hardware, without requiring an exascale system.**

## I. Introduction

The NAS Parallel Benchmarks (NPB) [1] were originally formulated as paper-and-pencil benchmarks that could be implemented in almost any way, so that they could take maximal advantage of the wide diversity of parallel programming models and computing systems that were available at the time of their inception. Shortly thereafter, as a convergence was taking place in such models and systems, source code versions of the NPB were made available by NASA, [2], first in MPI, and later in OpenMP, that were considered good parallel implementations. These have henceforth been used in procurements, to test software tools, and to guide system architecture for almost two decades. Some of their longevity can be attributed to the growing range of problem sizes – called *classes* by the NPB developers – which now reaches to class "F" for some of the benchmarks. While the NPB of the largest class provide a significant challenge to many current powerful HPC systems, they do not provide nearly enough work or concurrency to be run profitably on petascale or exascale systems.

Hence, we must extend the benchmarks beyond the predefined problem classes, which necessitates a rationale for scaling up the problem size. This does provide an opportunity to specify a certain balance between the problem size and the system resources. A popular scale-up rule, traditionally called *weak scaling* [3], [4], stipulates that the amount of work in the problem scales linearly with the available compute resources. We generalize the concept of weak scaling, and draw some inferences for what the generalizations imply for the trends of some important workload characteristics, namely the amount of computational work, the memory footprint, and the amount of data communicated. For many workloads these govern to first order the feasibility to run on a certain system based on the resources required, as well as their performance.

In this paper, we use the NPB *Block-Tridiagonal* (BT) workload and its MPI implementation as an example to illustrate how changing the size of an application under the different weak scaling rules impacts the computational work, the memory footprint and the communication volume. We derive from that analysis the feasibility of scaling, as well as the bottlenecks that are expected to dominate the performance at exascale. Finally, we propose solutions for reducing the impact of these bottlenecks. We note that BT and the *Scalar Pentadiagonal* (SP) benchmarks are functionally identical, and any analysis derived for BT qualitatively applies equally to SP.

The contributions of this paper include:

1) It proposes a novel use of analytical application models to derive application scaling behaviors for future systems. The method is described in detail in Sections IV to VII, and is illustrated through experiments.
2) It proposes a novel way of modifying certain MPI applications to run on a small number of ranks, yet accurately projecting performance to configurations with arbitrary numbers of ranks. Such a method allows us to study performance characteristics of applications on systems that are much larger than are available today.
3) It identifies scalability bottlenecks with the current BT MPI implementation and discusses alternative approaches to overcoming these bottlenecks.
4) It discusses how current architecture trends may affect the scalability of workloads in future systems. For example, memory capacity limitations may aggravate the communication latency problem for scaling to exascale systems.

The paper is organized in the following manner. Section II discusses and extends common methods used in scaling workloads. Section III provides a functional description of the NPB BT benchmark. Section IV describes a precise analytical

SC12, November 10-16, 2012, Salt Lake City, Utah, USA
978-1-4673-0806-9/12/$31.00 ©2012 IEEE

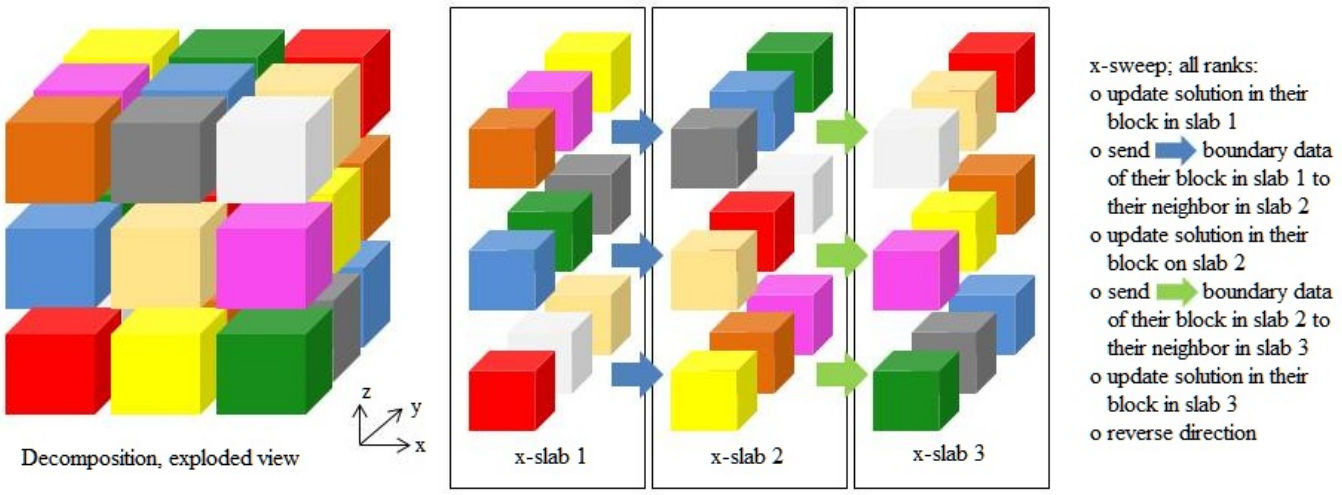
Decomposition, exploded view
z
y
x
x-slab 1
x-slab 2
x-slab 3
x-sweep; all ranks:
o update solution in their block in slab 1
o send boundary data of their block in slab 1 to their neighbor in slab 2
o update solution in their block on slab 2
o send boundary data of their block in slab 2 to their neighbor in slab 3
o update solution in their block in slab 3
o reverse direction

Fig. 1. **NPB BT multi-partition decomposition and algorithm, x-solver sweep only, for 9 MPI ranks. Blocks of same color belong to same rank. Y-solver (sweeps into plane of paper) and z-solver (sweeps from bottom to top) are executed similarly**

model based on code examination and discusses how we verify the communication part of the model. Section V describes an asymptotic analytical model that will be used for our scaling study. Section VI describes the scaling study and the expected behaviors of the various scaling methods. Section VII presents the experimental results of the scaling study, using our novel approach of emulating exascale systems. Section VIII discusses the impact of communication on scalability, as well as ways to reduce that impact. Section IX discusses the impacts of architecture trends on scaling workloads. Section X discusses related work, and Section XI concludes the paper.

## II. Background and Methodology

A number of computational problems exist today that could make use of the computation capability of an exascale system, in principle. However, the existing implementations of these problems may not be suitable for the exascale systems to be built. One approach to provide realistic projections of exascale workload behaviors is to scale up today's workloads. There are multiple ways of doing that. The first assumes that the current problem size of the workload is large enough and that the workload would not benefit further from increasing the resolution of the problem. For this type of workload, the additional compute resources provided by a larger system are devoted to obtaining the solution faster. This method of scaling is known as *strong scaling*. For the method to work, the problem must possess a high degree of parallelism and require little synchronization. No matter how much parallelism a workload has, the performance of such a workload under strong scaling will reach a point of diminishing returns once the system reaches a certain size. For NPB, the existing and the foreseeable problem sizes are too small to make sense for strong scaling on peta/exascale HPC systems. Therefore, strong scaling of these workloads is not considered in this paper.

A second way to scale workloads is to take existing workloads whose sizes are parameterized, and to scale them proportionally to the size of the system (iterations or time steps are usually not considered part of the workload size). This method of scaling is known as *weak scaling*. We distinguish between the following types of weak scaling. As the number of MPI ranks grows:

1) The amount of computational work per MPI rank remains constant.
2) The memory footprint per MPI rank remains constant.
3) The ratio between computational work and communication volume remains constant.

Weak scaling method 1 embodies what is traditionally called weak scaling [3], [5]. The amount of work per computational unit remains constant. As the number of computational units increases in a system, the size of the problem increases at the same rate.

Weak scaling method 2 is the same as weak scaling method 1 for problems where the amount of work per data element is constant. The NPB BT, SP, and LU benchmarks are examples of workloads that display this kind of behavior. However, other workloads, such as LINPACK or FFT, would have a different scaling behavior under weak scaling method 1 and 2, because the amount of work per data element in these workloads varies as the problem size changes.

Weak scaling method 3 requires that the problem size and the memory size per computational unit grow faster than the number of computational units, since higher communication costs are incurred per unit of computational work as the overall problem size grows. NPB BT and SP fall into this category. For LINPACK, on the other hand, weak scaling method 3 would actually allow the problem size to grow at a slower rate than the number of computational units in the system.

## III. Functional workload description

The BT workload is derived from a mature 3-dimensional *Computational Fluid Dynamics* (CFD) code, developed by NASA, and used at several research centers and in the aircraft industry. At its heart lies the Alternating Direction Implicit algorithm developed by Beam and Warming [6], which features strong data dependencies that change periodically during the course of the computation. This requires advanced parallelization techniques, especially on clusters, that provide good load balance while avoiding a massive amount of very fine-grain communications. The rest of the paper assumes that MPI is used as a programming model.

The technique chosen for BT is called *multi-partitioning*, which was first described by Bruno and Cappello [7], and subsequently by Naik et al. [8], and Van der Wijngaart [9]. The essence of the technique is to assign to each MPI rank a subset of the data space that consists of multiple distinct blocks (partitions), whose number is the square root of the total number of ranks. The partitions are arranged such that each slab of partitions normal to any of the three coordinate directions always contains exactly one partition of each rank. The leftmost panel in Figure 1 shows an example of the multi-partition decomposition for 9 MPI ranks. In this figure, each rank owns $\sqrt{9}$ (i.e. 3) blocks.

Although the algorithm has been described before in the above references, we repeat some of its salient features here for the sake of this paper's scaling analysis. The operations in each time step of the algorithm can be divided into four global phases.

1) *Right Hand Side* (RHS): computes a forcing term by evaluating a stencil operator on all grid points owned by the target rank independently. For points on the interior boundary of a grid block some data is required from neighboring blocks (communication) to evaluate the stencil operator.
2) *x-solve*: solves a set of block-tridiagonal linear equations – one for each grid line in the x-direction. This can be further subdivided into two steps, namely *Forward (Gaussian) Elimination* (FE) and *Back-Substitution* (BS). The FE step involves marching in the positive x-direction along each grid line and updating values assigned to each point on the grid line (array element). Whenever the computation on a grid line reaches the boundary of a block owned by the target rank, a communication is required to continue to the block owned by that rank's topological neighbor, see the middle panel of Figure 1. The BS step reverses the direction of traversal.
3) *y-solve*: same as x-solve, but in the y-direction.
4) *z-solve*: same as x-solve, but in the z-direction.

## IV. Precise Analytical Model

In this section we first describe the precise model of computation, memory usage, and communication for our target benchmark BT. Next, we verify the communication part of the model by comparing it against data we collected by running the actual BT benchmark.

TABLE I
BT: Number and size of messages

| Phase | Number of Messages | Size of Messages (bytes) |
|---|---|---|
| RHS | 6 | $80 * (q-1) * (N/q)^2$ |
| x-solve FE | $q-1$ | $240 * (N/q+1)^2$ |
| x-solve BS | $q-1$ | $40 * (N/q+1)^2$ |
| y-solve FE | $q-1$ | $240 * (N/q+1)^2$ |
| y-solve BS | $q-1$ | $40 * (N/q+1)^2$ |
| z-solve FE | $q-1$ | $240 * (N/q+1)^2$ |
| z-solve BS | $q-1$ | $40 * (N/q+1)^2$ |

TABLE II
BT: Computational work / rank / time step (flops)

| Phase | Computation (Flops) |
|---|---|
| RHS | $258 * N^3/q^2$ |
| x-solve FE | $q * 862 * (N/q)^3$ |
| x-solve BS | $q * 50 * (N/q)^3$ |
| y-solve FE | $q * 862 * (N/q)^3$ |
| y-solve BS | $q * 50 * (N/q)^3$ |
| z-solve FE | $q * 862 * (N/q)^3$ |
| z-solve BS | $q * 50 * (N/q)^3$ |
| Total (asymptotic) | $2994 * N^3/q^2$ |

### A. Model Description

Assume a cubical grid of $N^3$ points and a number of MPI ranks of $P = q^2$. According to the multi-partitioning technique described in section III, each rank receives responsibility for $q$ blocks of $(N/q)^3$ grid points each. The total communication surface area of all blocks owned by one rank is $6 * (q-1) * (N/q)^2$ grid points. The reason why the multiplication factor is $q-1$ and not $q$ is that communications only take place at interfaces between blocks, not on the outer boundary of the global grid. Because of the special arrangement of the grid blocks, each rank has the same number of grid block faces that lie on the outer boundary of the grid. Each MPI rank has exactly six logical communicating neighbors: $+x$, $-x$, $+y$, $-y$, $+z$, and $-z$, whose ranks can be computed statically, independent of the grid size.

Table I shows the number and sizes of the messages sent during one time step of the BT code. As mentioned in Section III, BT consists of four phases, namely RHS, x-solve, y-solve, and z-solve. Each of the directional solvers can be further subdivided into FE and BS phases, and we provide a cost breakdown according to this subdivision. It must be noted that the three directional solvers are identical to each other, except for the array dimension corresponding to the spatial direction on which they operate.

Table II lists the computations for one iteration of the BT loop in terms of aggregate floating point operations derived from source code inspection (to highest order only). This does not take into account the effects of vectorization, common subexpression elimination, occurrence of fused arithmetic operations, etc., which can change the coefficients of the polynomials in the table. However, the functional form of the

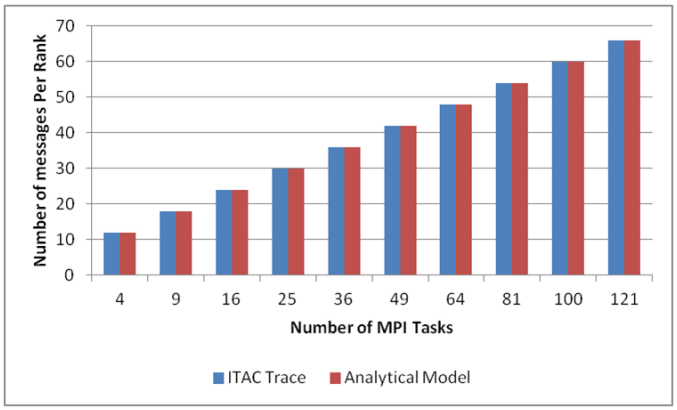


Fig. 2. **Number of messages (BT class C): Trace data (blue) vs Precise Model (red).**

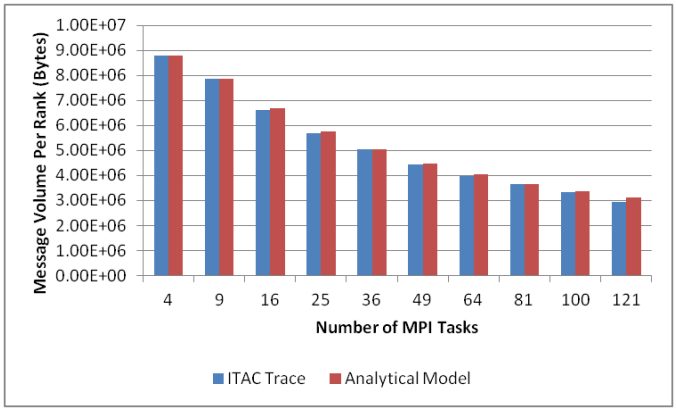


Fig. 3. **Message Volume (BT class C): Trace data (blue) vs Precise Model (red).**

polynomials is not affected by that.

The memory footprint per rank is given by: $280 * q * (N/q)^3 + 48 * q * ((N/q)+2)^3 + 40 * q * ((N/q)+4)^3$ bytes. This was obtained directly and accurately from the source code by computing the sizes of the major arrays.

### B. Communication Model Verification

We now describe the methodology used to verify the communication component of the precise model. We use the *Intel Trace Analyzer and Collector* (ITAC) tool to collect traces of the BT benchmark (from version 3.3) for different problem sizes (class A, B, and C), and compare the number of messages and the message volume from the application traces against the analytical model. Here we present the data pertaining to class C.

Figure 2 compares the total number of messages sent by each rank during one iteration of the main BT loop for the analytical model and the actual profile. We observe that model and measurement match each other exactly.

Figure 3 compares the total message volume (bytes) sent by each rank during one time step of the main BT loop for the analytical model and the actual profile. The analytical model and experimental data match exactly in a few cases. Any differences are due to the fact that the number of blocks in a certain coordinate direction does not always exactly divide the grid size in that direction. For the purpose of this investigation the differences are immaterial.

## V. Asymptotic Analytical Model

This section presents an *Asymptotic Model* that provides an approximate representation of the BT benchmark. It is derived from the precise model by only considering the higher-order terms. The asymptotic model is used in our scaling studies (Section VI).

At present we do not take into account the effect of message latency, which is reasonable for BT if the number of ranks is not very large. In Section VIII, we will relax this assumption and see how latency could impact scalability.

A simple analysis shows that for large block sizes ($N \gg q$) the workload parameters behave as follows:

- Computational work/rank/time step (flops): $2994 * N^3/q^2$
- Memory footprint/rank (bytes): $368 * N^3/q^2$
- Communication volume/rank/time step (bytes): $1320 * N^2/q$

## VI. Scaling BT

We now examine the effects of various types of weak scaling of BT, based on the results of Section V. As the number of ranks grows from $P$ to $P'$, $q$ will grow to $q' = \sqrt{P'/P}$.

Weak scaling method 1 dictates that the computational work remain constant. That means $2994 * N'^3/q'^2 = 2994 * N^3/q^2$. Solving for $N'$ yields $N' = N * \sqrt[3]{P'/P}$. Substituting this into the equations of Section V yields that the memory footprint remains constant, and that the data volume of communications grows by a factor of $\sqrt[6]{P'/P}$. We note that while the total number of grid points in the blocks owned by a certain rank (and hence the computational work carried out by that rank) is constant, the number of the blocks owned by that rank goes up. Hence, the number of points in each block shrinks, and the total surface area of the blocks belonging to a particular rank, which is proportional to the communication volume, goes up.

Weak scaling method 2 is equivalent to weak scaling method 1 for BT.

Weak scaling method 3 requires that the ratio between the computational work and the volume of communication per rank remain constant. This means $(N'^3/P')/(N'^2/\sqrt{P'}) = (N^3/P)/(N^2/\sqrt{P})$. Solving for $N'$ yields $N' = N * \sqrt{P'/P}$. Substituting this into the equations of Section V yields that the computational work, the memory footprint, and the communication volume all scale with a factor of $\sqrt{P'/P}$.

Figure 4 depicts the relationships graphically. The horizontal axis shows the number of ranks in the system. The vertical axis shows the growth per rank for each of the three metrics–the computational work, the memory footprint, and the amount of data communicated. Notice that in the case of weak scaling method 1 and 2, the amount of work and the memory footprint remain constant, but the communication volume increases slowly, at a rate of $\sqrt[6]{P'/P}$. Hence, for a million-fold increase in system size, there is a ten-fold increase in the communication volume. For weak scaling method 3, the

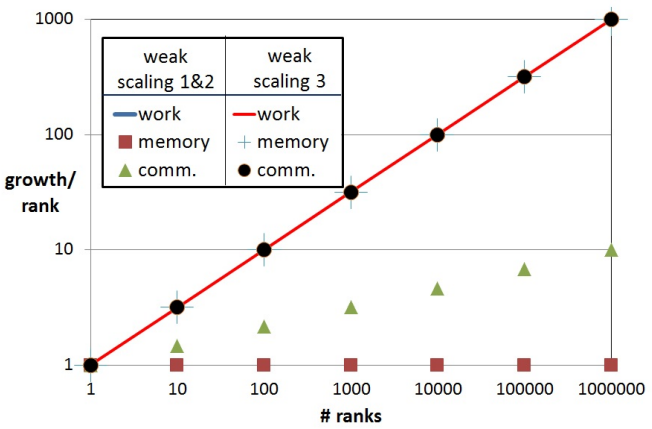


Fig. 4. **Comparing different types of weak scaling of NPB BT.**

amount of computational work, the memory footprint, and the amount of data communicated all grow at a rate of $\sqrt{P'/P}$. For the same million-fold increase in the number of ranks, all three metrics will grow by a factor of 1000. Compared to weak scaling method 1 and 2, method 3 requires much more resources. It is not practical to increase the memory per rank by a factor of 1000, and to wait for the result 1000 times longer than today. For example, a typical single-threaded run of BT Class C takes about 24 minutes on a single core of the Intel Core i7-975X processor. If we scale that workload to a system with a million such cores under weak scaling method 3 described in the paper, the run time will increase by a factor of 1000, to 400 hours (a little over 16 days). Hence, we reject weak scaling method 3 for BT.

## VII. Experimental results

We modify the original BT code provided by NASA to be able to emulate realistically the environment of a real exascale computer without requiring access to such a system. This is possible, because the particular arrangement of blocks assigned to each rank in the multi-partition method guarantees that any rank only needs to communicate with six topological neighbor ranks, regardless of the total number of ranks in the system and the total number of blocks assigned to each rank. The modified code leaves the entire code unchanged for MPI rank zero, including all the specifics of data structures needed for the multi-partition domain decomposition, all the relevant computations, and all communications. For that rank the program execution and communication granularity is indistinguishable from executing an exascale program on an exascale machine. The six remaining ranks, which we call *synthetic*, effectively only fulfill communication requirements for rank zero and do not communicate with any other ranks. They also skip some of the computations.

This configuration reasonably reflects the execution environment of a real exascale system with realistic network connectivity (such as a mesh or torus), because all communications are between topologically nearest neighbors, and can be scheduled such that never more than a very small number of network hops (just one for a 3-dimensional mesh or torus) needs to be traversed by any message. The method works because even though the synthetic ranks skip certain computations and communications, they do not produce any arithmetic exceptions themselves, nor do they trigger any on the target rank. In addition, control flow of the code is not data dependent, and the benchmark has a near-perfect load balance.

TABLE III
Emulated BT ranks and grid sizes (x-dimension only), weak scaling method 1.

| Emulated ranks | grid size | | |
|---|---|---|---|
| | class A | class B | class C |
| 16 | 161 | 257 | 408 |
| 64 | 256 | 408 | 648 |
| 256 | 406 | 647 | 1,028 |
| 1,024 | 645 | 1,028 | 1,632 |
| 4,096 | 1,024 | 1,632 | 2,592 |
| 16,384 | 1,625 | 2,590 | 4,114 |
| 65,536 | 2,580 | 4,112 | 6,531 |
| 262,144 | 4,096 | 6,528 | 10,368 |
| 1,048,576 | 6,501 | 10,362 | 16,458 |
| 4,194,304 | 10,321 | 16,449 | 26,125 |
| 16,777,216 | 16,384 | 26,111 | 41,471 |
| 67,108,864 | 26,007 | 41,450 | 65,832 |

We test our modified BT benchmark on a 360-node cluster running Intel MPI (version 4.0.3.008). Each node consists of two eight-core Xeon$^{TM}$E5-2670 processors with 64GB of memory. The nodes are connected in a fat-tree topology using Mellanox FDR Infiniband controllers (running OFED 1.5.3.1). MPI ranks within an SMP node communicate via shared memory.

Using the modified BT code, we emulate configurations ranging from 16 to 64 million ranks, representing small to exascale systems. Since we are interested in weak scaling, we use NPB classes A, B, and C, respectively, as baselines, and scale these up proportionally to the number of emulated ranks in the system, using weak scaling method 1. Table III shows the corresponding sizes of the overall grids for all the emulated ranks that were tested.

We also extend the model developed in Section V to extrapolate the 16-rank performance results for class A, B, and C analytically, using the following assumptions: computation time is constant, time spent on communications equals $K * L + V/BW$, where $K$ is the number of messages sent per rank, $L$ is the measured MPI latency between ranks, $V$ is the number of bytes sent between neighbors, and $BW$ is the measured MPI bandwidth for very large messages.

In Figure 5 we show the scalability of the emulated BT benchmark with classes A, B, and C as baselines, running on one SMP node under weak scaling method 1, as well as the corresponding curves for the extended model as described in the previous paragraph. The first observation is that the expected behavior of good scalability (efficiency above 75%) of all classes for sizable numbers of ranks (up to 1 million) is borne out by the emulation. Beyond that, we see that the

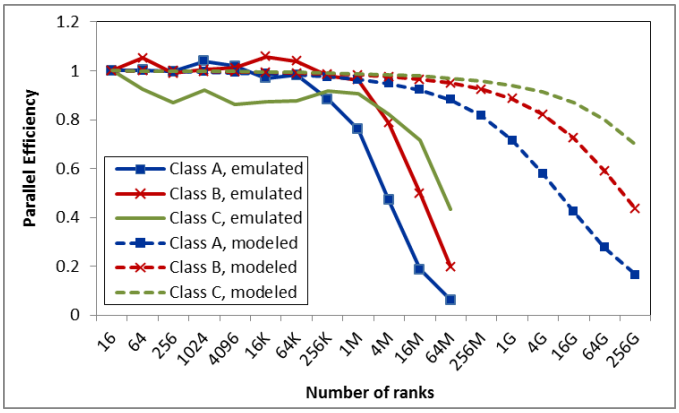


Fig. 5. **Performance results for weak scaling method 1 of BT with classes A, B, and C as baselines, using a shared memory node.**

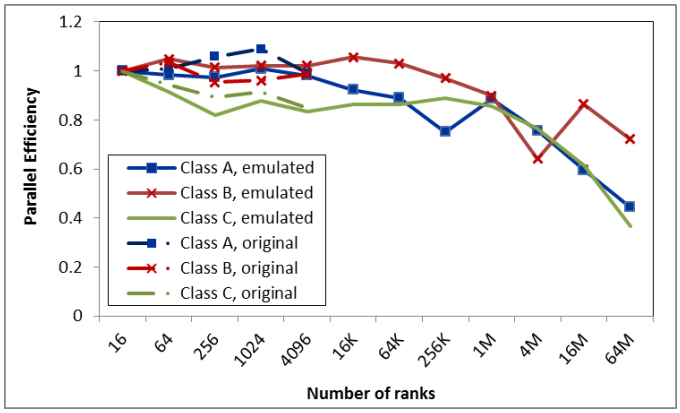


Fig. 6. **Performance results for weak scaling method 1 of BT with classes A, B, and C as baselines, using a distributed memory system.**

coarser granularity of class B allows it to scale further than class A, and class C scales further still, also as expected.

The second observation is that for classes A and B parallel efficiency actually exceeds 100% for certain modest numbers of ranks, even though the total data set size for all ranks remains unchanged as we scale up the problem size, in keeping with weak scaling method 1. We conjecture that the reason for this phenomenon is the fact that the multi-partition method implies a tiling of each rank's data set, which gets finer as the number of ranks increases. This improves the locality of memory accesses. However, once an individual partition's data fits inside the processor's cache, no more benefit is obtained from finer tiling, and loop overhead of the increasingly fine tiling actually starts to affect computational efficiency, as does the fact that decreased loop length renders prefetching and vectorization less efficient. Note that class C does not exhibit performance enhancement through tiling, due to its larger data set size.

The third observation is that the extended model, though qualitatively similar to the emulation results, shows a slower reduction in efficiency as the number of ranks increases. One reason for that is that the effect of reduced computational efficiency as the computational granularity decreases is not captured by the model. Another is that at high numbers of ranks the asymptotic formulas that we developed for computational and communication complexity do not hold anymore, since individual partitions become rather small. In addition, the linear formula for communication time we used is quite inaccurate for certain message sizes, which has an even more dramatic influence on performance of the modified code on an actual cluster. Finally, since the grid size is not evenly divisible by the number of partitions, some partitions are bigger than others in the same slab (see Figure 1), which means the code will suffer temporary load imbalances during the solver phases. For all these reasons we believe our emulation technique adds significant practical value to analytical modeling.

However, the analytical model does allow us to make a determination of the relative importance of the communication features of the benchmark. The model developed in Section VI predicts that communication overhead will eventually dominate performance, as it grows at a rate of $\sqrt[6]{P'/P}$. However, the reduction in the overall performance, both observed and modeled, for large numbers of ranks is much greater than can be explained by increased message volume alone. The explanation of this phenomenon lies in the fact that our model for weak scaling method 1 ignored message latency. This is fine for modest numbers of ranks, when the messages per rank are not so many and fairly large. As described in Section VI, the number of messages sent per time step scales asymptotically as $\sqrt{P'/P}$, so much faster than the message volume per rank. While the RHS phase always sends six messages (one to each of its neighboring ranks), the rest of the messages are sent during the three solver phases of the benchmark. The latencies associated with these messages can hardly be overlapped, because the moment a message is sent, its data are needed by another rank. Hence, these exposed latencies start to dominate performance at the largest scales. For example, at the very large end of the scale in Figure 5, the time due to latency is 20, 8, and 3 times larger than that due to bandwidth, and constitutes 80%, 50%, and 22% of total modeled execution time for classes A, B, and C, respectively. In that range aggregate modeled system performance is still well below an exaflop for all classes.

In Figure 6 we show the results of running the modified code, as well as of running the original code, on a cluster composed of the same SMP nodes that we used for the shared memory-based study. As can be observed, the emulation results are unexpected in terms of performance (large excursions), as well as in terms of scalability (class B scales better than class C). These results do not represent noise and are quite repeatable. We tracked down the source of the unexpected results to large variations in transmission times (up to an order of magnitude) on the Infiniband network of messages of particular modest sizes, using simple ping-pong tests. Hence, we show these results only for illustrative purposes, but do not provide any analysis.

## VIII. Overcoming latency overheads

We propose multiple different and–to some extent– complementary ways to decrease the impact of latency overheads. We plan to explore these solutions as part of our future work.

### A. Problem size increase

One effective way to reduce latency impact is to increase the amount of computational work per rank. Weak scaling method 3 is virtually guaranteed to provide the right amount, but is unaffordable. However, merely starting from a larger problem size and using weak scaling method 1 to scale it up already had a beneficial effect. This is evidenced by the improved scaling as we increase the baseline benchmark size from class A to B, and then to C, see the previous section.

### B. Bi-directional multi-partitioning

A creative way to hide latency in BT is to employ a variation of the method used in the NPB, called bi-directional multi-partitioning, described by Van der Wijngaart et al. in [10]. The key is that the forward elimination phases in the solver stages as described in Section III can be executed from opposite ends of the grid, working in alternating fashion from either side towards the middle of the grid, followed by a similarly alternating pattern in the back-substitution phases. The result is that the entire duration of the computation on a particular block can be used to overlap with message passing. Hence, as long as the size of a single message latency is less than the computational work on a block, it can be hidden completely.

### C. Hybrid parallelism

Another possibility is to adopt a hybrid OpenMP and MPI programming model to leverage the shared memory architecture of today's multi-core processors and reduce the amount and frequency of communications required over the physical network. Assuming the coherence protocol and on-die interconnect can effectively handle the on-die communications among the cores, the number of high-latency MPI communications would be reduced by a factor equal to the square root of the number of OpenMP threads per node. Assuming Moore's law still holds for many-core processor designs between now (2012) and 2018 (the time at which we expect to reach exascale workloads and systems), the number of cores would increase eight-fold. Using the hybrid OpenMP/MPI model, the latency cost could be reduced by a factor of $\sqrt{8}$ (i.e. 2.83). However, this model is not totally free, as OpenMP would introduce some thread fork/join and coherency protocol overheads. In addition, the assumption is that the amount of memory per core can be kept constant as the number of cores per node increases. Presumably, the benefit of reducing the MPI communication overhead will outweigh any other overheads introduced.

### D. Multi-zone algorithm

A more radical algorithm change to reduce the overhead of communication is to adopt the *multizone* approach, described in [11]. In this case the original NAS Parallel Benchmark formulation of BT is modified so that it solves the BT problem on a number of grids (*zones*) independently for each time step, after which some boundary values of the grids are exchanged. Consequently, the fine-grain communications required by the multi-partition method do not extend beyond the confines of any of the zones, and at the zone level only coarse-grain communications occur. It should be noted, however, that this changes the BT algorithm significantly, making it much less implicit, and hence slowing down its numerical convergence substantially.

## IX. Impacts of Architecture Trends

The previous section discusses the impacts of communication latency on scaling workloads. This section focuses on the implications of architecture trends on scaling the workloads.

To meet power constraints, the chip multiprocessor has become the design choice for current and future processors. The wide acceptance of this solution is evident from the number of current and planned multi/many-core commercial offerings. Moore's law continues to hold, showing a doubling of transistor counts every two years, which leads to a doubling of the number of cores per processor every two years, everything else staying equal. However, the memory interface is much limited by the number of pins available for the processor packages. Over the years, the size of processor packages has grown, but at a rate lower than the transistor density. As a result, the amount of memory bandwidth or capacity available to each core is getting smaller. Assuming that in the next six years the memory capacity grows by a factor of two, then the effective memory capacity per core drops by a factor of four. The implication of that is that the amount of data that a single MPI rank can work on is effectively one fourth of today's. This means that there will be less computation to do before any communication or synchronization. This finer granularity makes the problem more communication bound. This means that the latency hiding techniques we discussed in the previous section become more critical.

An architecture trend that is likely to have a positive impact on workload and system scaling is the integration of the network controller. As semiconductor manufacturing technology improves, more transistors can be integrated onto the same die. One way to make use of the increased transistor count is to integrate specific function units. Integration of the network controller can reduce communication latency; thus, it can positively impact the workload scaling discussed in this paper.

## X. Related Work

Workload scaling is an important and well studied area. There is a large body of prior work that studied the scalability of the NPB on high-performance computing systems. To name just a few, Ramachandran et al. presented a study of NPB on

the KSR-1 machine in [12], Wong et al. presented a study of the NPB on an SGI Origin 2000 machine in [13], and Saini et al. study the scalability of the NPB on NASA's Columbia system in [14]. These and other works mostly focused on understanding the strong scaling of the NPB on existing systems. While one can make reasonable projections based on existing measurements, our work extends this approach by integrating an in-depth analytical model of one of the benchmarks itself to allow accurate projection of computation, memory and communication behaviors, coupled with a method and rationale for increasing the workload size.

There is also a large body of work done on projecting workload performance for future systems. Two common approaches are based on extrapolating execution characteristics on existing systems and on analytical modeling. Snavely et al. use workloads characteristics (such as single processor performance and communication characteristics) to develop application models that can be used to predict how performance would change with system changes [4], [15]. Wu [16], Gonzalez [17], Zhai [18] developed novel frameworks that integrate system characteristics and simulations to further improve accuracy of the projections. Others include Kerbyson [19], Barnes [20], and McClanahan [21], who use analytical models to describe and predict application performance.

Our approach is similar to these efforts in that we also use an analytical model to predict application performance and architectural bottlenecks. However, our work differs from these previous works in that instead of solely using an analytical model to predict performance, our projection is supplemented with measurements of a modified version of the target benchmark on real systems. Our novel approach of modifying the BT benchmark preserves the execution behavior of the original benchmark, but only requires seven physical MPI ranks to emulate the execution of an arbitrary number of ranks. Measuring performance of the modified benchmark on real systems allows us to accurately predict how the BT benchmark will behave if we apply weak scaling to a system with similar processors.

## XI. Summary, Conclusions, and Future Work

This paper studies the various methods that can be used to scale workloads to exascale. Using the NPB BT benchmark as an example, we study the implications of different methods of weak scaling on the memory footprint, the amount of computational work, and on communication volume. To verify our models, we modify the BT code to allow us to emulate systems of arbitrary size, ranging from small to exascale, using a technique called synthetic ranks. Our experimental results show that traditional weak scaling of the BT class A workload allows us to scale to a system with one million ranks while keeping the scaling efficiency at 75% or higher. Using class B or C as the initial starting point, we can scale to 4M and 16M ranks while keeping the same scaling efficiency. Scaling beyond these points shows a rapid drop in efficiency, due among others to an increase in communication latency costs. Several methods can be used to mitigate the effect of increased latency overheads, ranging from increasing problem size or changing algorithm (bi-directional multi-partitioning or multi-zone), to introducing a hybrid OpenMP/MPI programming model. In reviewing the current trends of architecture changes, we also found that reduction in memory capacity per computational unit is likely to aggravate the impact of communication latency. The integration of a network controller may be a partial solution to such a problem.

We plan to generalize our scaling methods and our synthetic ranks technique to other workloads, starting with some of the other NPB. The current success of the synthetic ranks technique derives from the fact that BT carries out virtually exclusively topologically nearest-neighbor communications. The *Scalar Penta-diagonal* (SP), *Multigrid* (MG) and *Lower-Upper Gauss-Seidel* (LU) NPB workloads fall into that same category. However, we do not expect good performance for LU at very large system sizes; it features a software pipeline parallelization technique whose inevitable load imbalances induced by pipeline fill and drain will hamper scalability ever more severely as the system size grows. The *Fast Fourier Technique* (FT) and *Conjugate Gradient* (CG) NPB workloads feature substantial non-local communications, whose extent grows as the system size grows. Hence, they are also not expected to be emulated very well on small systems and using synthetic ranks. They will also have difficulty scaling well to very large system sizes.

## References


[1] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, "The nas parallel benchmarks," The International Journal of Supercomputer Applications, Tech. Rep., 1991.

[2] D. Bailey, H. T., S. W.C., R. Van der Wijngaart, A. Woo, and M. Yarrow, "The nas parallel benchmarks 2.0," NAS Systems Division, NASA Ames Research Center, Moffett Field, CA, Tech. Rep. NAS-95-020, October 1995.

[3] G. J.L., "Reevaluating amdahl's law," *Communications of the ACM 31(5)*, pp. 532–533., 1988.

[4] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha, "A framework for performance modeling and prediction," in *Supercomputing, ACM/IEEE 2002 Conference*. IEEE, 2002, pp. 21–21.

[5] S. J. Culler, D. and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*, 1st ed. Morgan Kaufmann, 1998, the Morgan Kaufmann Series in Computer Architecture and Design.

[6] B. R. and W. R.F, "An implicit finite-difference algorithm for hyperbolic systems in conservation law form," *Journal of Computational Physics, 22*, pp. 87–110, 1976.

[7] J. Bruno and P. Cappello, "Implementing the beam and warming method on the hypercube," in *Proceedings of the third conference on Hypercube concurrent computers and applications-Volume 2*. ACM, 1989, pp. 1073–1087.

[8] N. Naik, V. Naik, and M. Nicoules, "Parallelization of a class of implicit finite difference schemes in computational fluid dynamics," *International Journal of High Speed Computing, Vol. 5, No. 1*, pp. 1–50, 1993.

[9] R. Van der Wijngaart, "Efficient implementation of a 3-dimensional adi method on the ipsc/860," in *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*. IEEE/ACM, 1993.

[10] R. F. Van Der Wijngaart, M. Yarrow, and M. H. Smith, "An architecture-independent parallel implicit flow solver with efficient i/o," in *8 th SIAM Conf. Parallel Proc. for Scientific Computing*, 1997.

[11] R. Van der Wijngaart and H. Jin, "Nas parallel benchmarks, multi-zone versions," NAS Division, NASA Ames Research Center, Moffett Field, CA, Tech. Rep. NAS-03-010, 2003.

[12] U. Ramachandran, G. Shah, S. Ravikumar, and J. Muthukumarasamy, "Scalability study of the ksr-1," *Parallel Computing*, vol. 22, no. 5, pp. 739–759, 1996.

[13] F. Wong, R. Martin, R. Arpaci-Dusseau, and D. Culler, "Architectural requirements and scalability of the nas parallel benchmarks," in *Supercomputing, ACM/IEEE 1999 Conference*. IEEE, 1999, pp. 41–41.

[14] S. Saini, J. Chang, R. Hood, and H. Jin, "A scalability study of columbia using the nas parallel benchmarks," *Journal of Comput. Methods in Sci. and Engr*, 2006.

[15] D. Bailey and A. Snavely, "Performance modeling: Understanding the past and predicting the future," *Euro-Par 2005 Parallel Processing*, pp. 620–620, 2005.

[16] X. Wu and F. Mueller, "Scalaextrap: Trace-based communication extrapolation for spmd programs," *SIGPLAN Notices*, vol. 46, no. 8, p. 113, 2011.

[17] J. Gonzalez, J. Gimenez, M. Casas, M. Moreto, A. Ramirez, J. Labarta, and M. Valero, "Simulating whole supercomputer applications," *Micro, IEEE*, vol. 31, no. 3, pp. 32–45, 2011.

[18] J. Zhai, W. Chen, and W. Zheng, "Phantom: predicting performance of parallel applications on large-scale parallel machines using a single node," in *ACM Sigplan Notices*, vol. 45, no. 5. ACM, 2010, pp. 305–314.

[19] D. Kerbyson, H. Alme, A. Hoisie, F. Petrini, H. Wasserman, and M. Gittings, "Predictive performance and scalability modeling of a large-scale application," in *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*. ACM, 2001, pp. 37–37.

[20] B. Barnes, B. Rountree, D. Lowenthal, J. Reeves, B. de Supinski, and M. Schulz, "A regression-based approach to scalability prediction," in *Proceedings of the 22nd annual international conference on Supercomputing*. ACM, 2008, pp. 368–377.

[21] C. McClanahan, K. Czechowski, C. Battaglino, K. Iyer, P. Yeung, and R. Vuduc, "Prospects for scalable 3d ffts on heterogeneous exascale systems," 2011.